An Industry-Standard Benchmark Consortium

OABench™ Version 2.0

Benchmark Name: Text (Text Parsing)

Highlights

- **Benchmarks potential performance of a printer interpretive control language**
- **Parses Boolean expressions made up of text strings**
- **Tests bit manipulation, comparison, and indirect reference capabilities.**
- **Largely shift/rotates with integer math and logical compares/branches**
- **A component of the EEMBC OAV2mark™**
- **Three data files**
- **Implements cyclical redundancy checksum (CRC) for self-checking**

History, Application and Restrictions

The Text (Text Parsing) Benchmark is representative of a printer application where an interpretive control language like PCL or PostScript is parsed. The algorithm parses Boolean expressions represented as text lines made up of variables, constants, and operators. The variables are space separated words, from 1 to 64 characters long, the constants are single character “T” or “F” and the operators may either be single-character symbols (& | !) or their phonetic equivalents (and, or, not). Standard precedence rules for expression parsing apply.

Benchmark Description

Input to the benchmark consists of rule data files that are loaded via the EEMBC RAMfile system. OABench Version 2 has four dataset files, one of which is used for profiling, compared with just one for OABench Version 1.1. These files are found in the libtxt directory. Much of the code is generated by the cheader subsystem in Version 2. The strings consist of variables, constants, and operators separated by spaces. For example:

"sss and fred implies (red & blue) or fred"

The expression is broken down into a binary tree structure, with each branch on the tree being an operand (a single variable, or a constant, or a reference to yet another tree node representing another expression). Unary operators are stored as modifiers to each of the branches. The resulting structure is then traversed to evaluate the value of the expression.

The benchmark avoids calling a memory allocation routine by statically declaring and managing a 1,000-node buffer. After the timed iterations have been completed, the test is run one additional time and a CRC is calculated for the binary tree to be used for checking for correct operation.

For each line, the benchmark breaks the expression into a binary tree structure, where each node contains a binary expression with two operands (each with a possible unary operator) and a binary operator. The operands may be variables, constants, or pointers to further nodes which themselves

represent binary operations, etc.

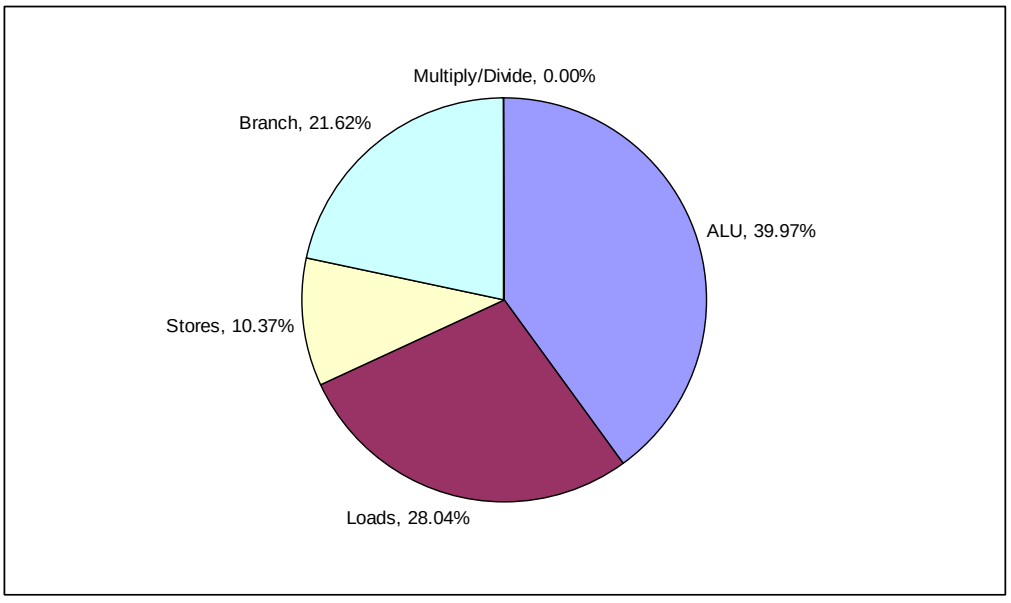
A debug mode is provided (in this case, the #define BENCHMARK) to activate the main timed loop, controlled by the test harness. If this is not defined, then the program goes into an interactive mode where each "rule" (Boolean expression) is entered by the user, parsed, and then printed out as a truth table

If the program is in debug mode (i.e., not in benchmark mode), then the program evaluates the expression for all possible values of the variable list. This is done with a recursive function to set the variables, and then by evaluating the expression stored in the binary tree.

1,000 iterations are the default, and 1,000 for CRC verification runs.

Analysis of Computing Resources

This benchmark exercises the byte manipulation, pointer comparison, indirect reference handling and stack manipulation capabilities of a processor.



The instruction mix is shown in the pie chart. The percentages may vary across architectures. The C library functions strcmp () and strncpy() are used extensively by this benchmark, and a well-designed and optimized C library would improve performance. Unlike other EEMBC benchmarks, dynamic memory allocations via malloc() are avoided. No floating-point calculations are used. The code size and the data size are moderate.

Optimizations Allowed

Out of the Box / Standard C
Full Fury / Optimized

- The C code must not be changed for Out-of-the-Box unless it must be modified to get it to compile. All changes must be documented, authorized by the certification authority, and must not have a performance impact.
- For Out-of-the-Box, additional hardware can be used if it does not require code changes.
- All optimized libraries must be part of the standard compiler package, and/or available to all customers
- The EEMBC Test Harness Lite must be used. Test harness changes may be made for portability reasons if they do not impact performance.
- For Optimized, the basic algorithm may not be changed, but the code may be rewritten in assembler. Re-writing the code to take advantage of parallelism is allowed so long as the correct answers are achieved using any arbitrary keys (not just those supplied in the benchmark code).
- For Optimized, optimized libraries can be used if they are publicly available.
- For Optimized, in lining is allowed.
- Additional data files may be used during certification to ensure the correctness of the optimized benchmark. You should ***not*** assume data patterns during optimization.
- Profile directed optimization is allowed using training data set 1, ruledata1.txt.